Fast Magnetic Tape Utilities for VAX/VMS Computers

N. E. Olson, R. F. Jurgens, and J. L. Robinett
Communications Systems Research Section

A package of subroutines for VAX/VMS computers has been developed to simplify the use of the QIO (Queued Input/Output) interface with magnetic tape drives. Routines are provided to perform all of the most common tape functions.

I. Introduction

As part of the standard software for their VAX/VMS computers, the Digital Equipment Corporation provides the SYS$QIO and SYS$QIOW routines to enable a program to request low-level operations on many I/O devices. Unfortunately, because of the general nature of these routines, the parameters they demand are often much more complex than would be necessary for simple operations such as rewinding a magnetic tape drive. In addition, there are several confusing deviations from the standard methods of argument binding. In order to decrease the frequency of programming errors and improve programmer productivity, we have written MTUTILS, a package of magnetic tape utilities which provide most of the flexibility of the QIO interface for tape operations with a maximum of simplicity. These routines use only the standard FORTRAN and Pascal methods of argument binding and have comparatively simple argument lists.

Table 1 contains a summary of the major routines. In addition to the source code for the routines, the package includes a help file for use with the VAX/VMS on-line help facility, a command procedure to build and install the software, and a FORTRAN include-file and a Pascal environment file to define the structures used by the package.

The routines can be divided into two groups: one which provides basic tape operations, and one which allows certain operations to be performed on a "list of tape drives." This last group is meant for use with datasets which are too large to be stored on a single magnetic tape. It allows the programmer to define a sequence of tape drives which will be written or read in cyclic order, the next drive in the sequence being automatically engaged when the current drive encounters the end of its tape.

II. Usage

The actual caller interface for each routine is listed in the appendix. Below is a description of the sample FORTRAN program shown in Fig. 1. It uses the MTUTILS package to write a record to tape. The file "MTUTILS. ICL" is the include-file which must be included in all FORTRAN routines that use the MTUTILS package. Note the variable called DSCR in the variable declaration section. It is a descriptor block filled by the INIT_UNIT routine and used by the other routines to store information about the tape unit. The first thing the program does is enable the diagnostics. These are informational and error messages which, if requested, are written to the logical file name FOR$PRINT. After getting the user's

input, the program calls the TRANSLATE_DENSITY routine to translate the density in bytes per inch into the density code required by INIT_UNIT. As with most of the routines, an error flag is returned indicating whether its function was successfully performed. If this flag is set, the main program can call several status-checking routines to obtain more detailed information about the error.

Next, the program calls INIT_UNIT, which performs several functions such as mounting and allocating the drive and assigning it a channel, as well as filling the descriptor. Now that the descriptor has been filled, it can be passed to the other routines which perform the following functions: read a record, write a record, write an End-Of-File mark, rewind the tape drive, skip files (forward or reverse), skip records (forward or reverse), search for the End-Of-Volume mark, or release the unit. (For a list of tape units, as opposed to a single unit, the following exceptions apply: rewind cannot be done, and neither files nor records can be skipped in the reverse direction.) Also provided are routines that will make asynchronous requests to read a record, write a record or rewind a tape drive. After making such a request, the program can proceed to do some computation, and when it needs to know that the request has been satisfied, it can call the WAIT-READY routine, which waits until this occurs.

Finally, the sample program writes the record to tape, prints out a message to that effect, and calls RELEASE_UNIT to perform clean-up functions such as dismounting and deallocating the unit, with the option specified to automatically unload the tape as well.

The program does not take measures to recover from errors, but the MTUTILS package provides several facilities to do so. In addition to requesting diagnostic messages, the program can obtain status flags and other data concerning the tape drive. For example, in the sample program, the length of the last record read or written is retrieved using the GET_LENGTH function. In case of more obscure error conditions, the program can get the actual system status word from the most recent system service call by using the GET_STATUS function. The most commonly checked flags are included as parameters to the relevant routines (e.g., WRITE_TAPE automatically returns an end-of-tape flag). The help file which comes with the package contains complete details about all of the routines.

III. Performance

The read and write routines have been timed with a TU-78 tape drive. The following formulas give a good estimate of the time needed to read or write a record of L bytes on a lightly loaded VAX for the particular tape drive on which the routines were timed (these times, of course, will be too optimistic for a heavily loaded VAX). TU-78 tape drives can operate at a density of either 1600 or 6250 bytes per inch (bpi). At 1600 bpi, the time in milliseconds per record is given by:

$$T = 0.00537L + 13.3$$

At 6250 bpi, the time is given by:

$$T = 0.00138L + 10.7$$

These figures indicate a burst rate of about 116 inches per second (ips) for our tape drive. The nominal rate for TU-78 drives is 125 ips.

Table 1. Summary of the major routines[a]

Operation[b]	Versions Included			
	For single tape drives		For a list of tape drives	
	Synchronous	Asynchronous	Synchronous	Asynchronous
Initialize unit	X		X	
Release unit	X		X	
Read record	X	X	X	X
Write record	X	X	X	X
Write EOF mark	X		X	
Rewind	X	X		
Skip files forward	X		X	
Skip files reverse	X			
Skip records forward	X		X	
Skip records reverse	X			
Search for EOV mark	X		X	
Wait for completion (of asynchronous request)	X		X	

[a]See the appendix for a complete list including status tests, etc.
[b]EOF means End Of File; EOV means End Of Volume.

```
*************************************************************************
*   This program writes a record of zeroes as the first record on a     *
*   tape.  The user specifies the record length, the name of the        *
*   tape drive and the tape density.                                    *
*                                                                       *
*   If a tape error occurs, an error message is printed and the         *
*   program halts.                                                      *
*                                                                       *
*   This program is not bullet-proof, e.g. the record length is not     *
*   tested to be within bounds.  This allows the user to force some     *
*   error messages out of the MTUTILS diagnostic facility.              *
*************************************************************************

        INCLUDE 'MTUTILS.ICL'           !This program uses the MTUTILS package

        PARAMETER (MAX_RECORD_LEN = 65535)

        LOGICAL ERROR_FLAG, EOT_FLAG    !EOT = End Of Tape
        INTEGER DENSITY
        INTEGER FORMAT
        DATA FORMAT /0/                 !Flag to use the default tape format
        INTEGER LENGTH
        BYTE BUFFER(MAX_RECORD_LEN)
        DATA BUFFER /65535*0/           !Initialize BUFFER to all zeroes

        CHARACTER*10 TAPE               !Name of the desired tape drive
        RECORD /TAPE_DSCR/ DSCR         !Declare a tape descriptor


C  Enable printing of diagnostic and error messages
C  (DIAGS is a constant defined in the INCLUDE file)

        CALL SET_DIAGNOSTICS(DIAGS)

C  Get the user's specifications

        WRITE(*,*), 'Enter the length of the record to be written (in bytes): '
        READ(*,*), LENGTH

        WRITE(*,*), 'Enter the name of the desired tape unit: '
        READ(*,10), TAPE
10      FORMAT(A10)

        WRITE(*,*), 'Enter the desired tape density (bpi): '
        READ(*,*), DENSITY

C  Translate the density in bytes per inch into a tape density code

        CALL TRANSLATE_DENSITY (DENSITY,ERROR_FLAG)
        IF (ERROR_FLAG) STOP    !Diagnostics have already been printed

C  Initialize the unit

        CALL INIT_UNIT (TAPE,DENSITY,FORMAT,DSCR,ERROR_FLAG)
        IF (ERROR_FLAG) STOP    !Diagnostics have already been printed

C  Write the record

        CALL WRITE_TAPE (DSCR,LENGTH,BUFFER,RETRY,EOT_FLAG,ERROR_FLAG)
        IF (ERROR_FLAG) STOP    !Diagnostics have already been printed

C  Print out the length of the last record written

        WRITE(*,*), 'A record of length ', GET_LENGTH(DSCR)
        WRITE(*,*), 'has been written.'
        IF (EOT_FLAG) WRITE(*,*), 'The end of the tape was reached.'

C  Release the unit and unload the tape.

        CALL RELEASE_UNIT (DSCR,UNLOAD,ERROR_FLAG)
        IF (ERROR_FLAG) STOP    !Diagnostics have already been printed

        END
```

Fig. 1. Sample program

Appendix
Summary of FORTRAN Routines

A summary of the FORTRAN caller interfaces to each of the routines in the package is provided in this appendix.

```
Routines with plural names (e.g. INIT_UNITS and READ_TAPES) and
those whose names begin with "LIST_" are for a list of tape units.
Others apply to single tape units.  Routines whose names begin
with a "Q" perform asynchronous requests.
After the list of routines is a description of their parameters.


Initialization and release:
 CALL TRANSLATE_DENSITY(DENSITY, ERROR_FLAG)
 CALL SET_DIAGNOSTICS(DIAG_FLAG)
 CALL INIT_UNIT(NAME, DENSITY, FORMAT, DSCR, ERROR_FLAG)
 CALL INIT_UNITS(NUMUNITS, NAMES, DENSITY, FORMAT, DLIST, ERROR_FLAG)
 CALL RELEASE_UNIT(DSCR, UNLOAD_FLAG, ERROR_FLAG)
 CALL RELEASE_UNITS(DLIST, UNLOAD_FLAG, ERROR_FLAG)


Reads and Writes:
 CALL QREAD_TAPE(DSCR, BYTE_COUNT, BUFFER, RETRY_FLAG, ERROR_FLAG)
 CALL QREAD_TAPES(DLIST, BYTE_COUNT, BUFFER, SWAPNUM, RETRY_FLAG, ERROR_FLAG)
 CALL READ_TAPE(DSCR, BYTE_COUNT, BUFFER, RETRY_FLAG, EOV_FLAG, ERROR_FLAG)
 CALL READ_TAPES(DLIST, BYTE_COUNT, BUFFER, SWAPNUM, RETRY_FLAG, EOV_FLAG, ERROR_FLAG)
 CALL QWRITE_TAPE(DSCR, BYTE_COUNT, BUFFER, RETRY_FLAG, ERROR_FLAG)
 CALL QWRITE_TAPES(DLIST, BYTE_COUNT, BUFFER, SWAPNUM, RETRY_FLAG, ERROR_FLAG)
 CALL WRITE_TAPE(DSCR, BYTE_COUNT, BUFFER, RETRY_FLAG, EOT_FLAG, ERROR_FLAG)
 CALL WRITE_TAPES(DLIST, BYTE_COUNT, BUFFER, SWAPNUM, RETRY_FLAG, ERROR_FLAG)
 CALL WRITE_EOF(DSCR, ERROR_FLAG)
 CALL LIST_WRITE_EOF(DLIST, SWAPNUM, ERROR_FLAG)
```

```
Skipping and Searching:

 CALL QREWIND(DSCR, UNLOAD_FLAG, ERROR_FLAG)

 CALL REWIND(DSCR, UNLOAD_FLAG, ERROR_FLAG)

 CALL SKIP_FILES(DSCR, COUNT, ERROR_FLAG)

 CALL LIST_SKIP_FILES(DLIST, COUNT, SWAPNUM, ERROR_FLAG)

 CALL SKIP_RECORDS(DSCR, COUNT, ERROR_FLAG)

 CALL LIST_SKIP_RECORDS(DLIST, COUNT, SWAPNUM, ERROR_FLAG)

 CALL SEARCH_EOV(DSCR, EOT_FLAG, ERROR_FLAG)

 CALL LIST_SEARCH_EOV(DLIST, EOT_FLAG, SWAPNUM, ERROR_FLAG)


Synchronizers:

 CALL WAITREADY(DSCR, ERROR_FLAG)

 CALL LIST_WAITREADY(DLIST, SWAPNUM, ERROR_FLAG)


Routines to extract extra information and error conditions:

 LENGTH = GET_LENGTH(DSCR)        LENGTH = LIST_GET_LENGTH(DLIST)

 STATUS = GET_STATUS(DSCR)        STATUS = LIST_GET_STATUS(DLIST)

 BOT_FLAG = TEST_BOT(DSCR)        BOT_FLAG = LIST_TEST_BOT(DLIST)

 EOF_FLAG = TEST_EOF(DSCR)        EOF_FLAG = LIST_TEST_EOF(DLIST)

 EOT_FLAG = TEST_EOT(DSCR)        EOT_FLAG = LIST_TEST_EOT(DLIST)

 EOV_FLAG = TEST_EOV(DSCR)        EOV_FLAG = LIST_TEST_EOV(DLIST)

 HWL_FLAG = TEST_HWL(DSCR)        HWL_FLAG = LIST_TEST_HWL(DLIST)

 LOST_FLAG = TEST_LOST(DSCR)      LOST_FLAG = LIST_TEST_LOST(DLIST)

 PAR_ERR_FLAG = TEST_PAR_ERR(DSCR)

 PAR_ERR_FLAG = LIST_TEST_PAR_ERR(DLIST)

 CURRENT_DRIVE = GET_UNIT_NUM(DLIST)

 CALL ERROR_CHECK(DSCR, BOT_FLAG, EOF_FLAG, EOT_FLAG, HWL_FLAG,
                  LOST_FLAG, PAR_ERR_FLAG)

 CALL LIST_ERROR_CHECK(DLIST, BOT_FLAG, EOF_FLAG, EOT_FLAG, HWL_FLAG,
                       LOST_FLAG, PAR_ERR_FLAG)
```

Below is a table explaining the parameters used in this package.

Note: Non-standard types such as "word" and "TAPE_LIST" are defined in the include file (for FORTRAN) or the environment file (for Pascal)

Parameter	FORTRAN type	Pascal type	Description
BOT_FLAG	LOGICAL	boolean	Beginning Of Tape
BUFFER	BYTE or INTEGER	integer	Data record buffer
BYTE_COUNT	INTEGER*2	word	Length of record
COUNT	INTEGER*2	word	Number of records or files
DENSITY	INTEGER	integer	Tape density
DIAG_FLAG	LOGICAL	boolean	Controls diagnostic messages
DLIST	RECORD/TAPE_LIST/	TapeList	List of descriptors
DSCR	RECORD/TAPE_DSCR/	TapeDscr	Tape descriptor
EOF_FLAG	LOGICAL	boolean	End Of File
EOT_FLAG	LOGICAL	boolean	End Of Tape
EOV_FLAG	LOGICAL	boolean	End Of Volume
ERROR_FLAG	LOGICAL	boolean	Error
FORMAT	INTEGER	integer	Tape format
HWL_FLAG	LOGICAL	boolean	Hardware Write Lock
LOST_FLAG	LOGICAL	boolean	Tape position Lost
NAME	CHARACTER*L	packed array of char	Logical name of tape unit
NAMES	CHARACTER*L(N)	array of Name	List of logical unit names
NUMUNITS	INTEGER	integer	Length of name-list
PAR_ERR_FLAG	LOGICAL	boolean	Parity Error
RETRY_FLAG	LOGICAL	boolean	Controls automatic error-retry
SWAPNUM	INTEGER	integer	If non-zero, tape to swap*
UNLOAD_FLAG	LOGICAL	boolean	Controls unloading of tape

* Number of the tape unit in a tape list which has reached EOT and needs to be swapped for the next tape in the volume set. This value is returned by the list-of-tape routines so that the calling program can alert the operator when tape swaps are necessary.